REMARKS

The Specification has been amended for clarification purposes only and, therefore, does not present new matter. No amendments, cancellations or additions have been made to the present claims. Therefore, claims 1-17 are currently pending in the case. Further examination and reconsideration of the presently claimed application are respectfully requested.

Section 103 Rejections

Claims 1-17 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Introducing Swing*, written by Sun (hereinafter "IS-SUN") and *Mixing heavy and light components*, written by Amy Fowler (hereinafter "M-SUN"). To establish a *prima facie* obviousness of a claimed invention, all claim limitations must be taught or suggested by the prior art. *In re Royka*, 490 F.2d 981, 180 U.S.P.Q. 580 (C.C.P.A. 1974); MPEP 2143.03. Obviousness can only be established by combining or modifying the teachings of the prior art to produce the claimed invention where there is some teaching, suggestion, or motivation to do so. *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed.Cir. 1988); *In re Jones*, 958 F.2d 347, 21 USPQ2d 1941 (Fed. Cir. 1992); and, MPEP 2143.01. The cited art does not teach or suggest all limitations of the currently pending claims, some distinctive limitations of which are set forth in more detail below.

None of the cited art teaches or suggests a system, method, or computer-readable storage device for graphical display of an object, where the system comprises a peer component, which is adapted to receive events pertaining to the object and route the events to a proxy component, which then associates the object with a graphics resource component and invokes methods thereof to display the object. Independent claim 1 states:

> A system for graphical display of an object created by an application program running under an operating system, comprising: a graphics resource component adapted to display the object independently of the operating system; a proxy component, which associates the object with the graphics resource component and invokes methods of the graphics resource component to display the object; and a peer component, adapted to receive events pertaining to the object and route the events to the proxy component.

Independent claims 9 (i.e., a method for graphical display of an object) and 17 (i.e., a computer-readable storage device) recite similar limitations.

In particular, the presently claimed case enables an application program (e.g., a Java application) to be truly portable across all operating system (OS) platforms by providing a means for maintaining the look and feel of the application program's graphical user interface (GUI). "A software program is said to be 'portable' across various platforms if the program can run without modification on any of those platforms." (Specification, page 6, lines 23-24). Before such means are disclosed, the presently claimed case highlights several drawbacks of prior attempts at portability.

For example, Java applications traditionally utilize an application program interface (API), known as the abstract windowing toolkit (AWT), which uses heavyweight software components (written in native code) to display images in a manner dependent on the particular OS running the API. Unfortunately, certain limitations of the AWT prohibit portability of application programs, especially in terms of the look and feel of the API. *See*, e.g., Specification, page 7, line 24 to page 9, line 10, and page 18, line 10 to page 19, line 23. In an effort to overcome the platform-dependency of the AWT, Swing was developed as part of the Java Foundation Classes (JFC) to assist programmers in developing rich-featured applications. An API written using Swing contains no native code, and therefore, can be run on any OS by borrowing or "inheriting" the native code from other software components (e.g., from heavyweight AWT components). *See*, e.g., Specification, page 9, lines 14-30, and page 19, line 25 to page 20, line 4.

The lightweight software components of Swing cannot be directly used to eliminate the platform-dependency of the user interface in applications that use AWT. Since Swing versions of many AWT components (e.g., containers, such as frames) are unavailable, many programmers have attempted to mix Swing and AWT components within a given API. *See*, e.g., Specification, page 21, lines 1-13. However, straightforward mixing of Swing and AWT components tends to create many problems that programmers have simply learned to accept. For example, a programmer may attempt to add a Swing component to an existing (AWT) frame layout, with the intention of painting over the platform-dependent AWT renderings of objects in the layout. However, the lightweight Swing components cannot successfully compete with the pre-existing heavyweight AWT components, since the heavyweight AWT components will obscure any overlapping lightweight components. Other problems arise when programmers attempt to substitute an equivalent Swing control for the original AWT control, and when attempts are made to directly substitute heavyweight Peers (which invoke graphics resources of the operating system to render objects) for lightweight Peers (which utilize lightweight Swing objects for rendering objects). *See*, e.g., Specification, page 22, lines 1-28.

The presently claimed case provides a system, method and computer-readable storage device for producing a truly portable, platform-independent API that overcomes the difficulties associated with migrating application programs from AWT to Swing. In general, the presently claimed case may be considered a functional extension of Swing – referred to as "AWTSwing" – since the platform-independent features of Swing are made available to AWT-based Java applications without modifying the AWT-based application. *See*, e.g., Specification, page 24, lines 8-29.

In the embodiment of FIG. 8, the present system and method creates a proxy object (e.g., JButton proxy 88) for associating a heavyweight object (e.g., Button object 30) with lightweight graphic resource components. The proxy object may then invoke methods (contained, e.g., within JButton object 48) of the lightweight graphic resource components to display an image of an object (e.g., a button control) within a heavyweight container (e.g., within AWT frame 41). The object is displayed independently of the operating system running the API. In addition, a lightweight peer component (e.g., JComponent peer 84) of the system is configured to receive events pertaining to the heavyweight object (e.g., a mouse click at the button control), and to route the received events to the proxy component. In this manner, the presently claimed case enables lightweight peers to be indirectly substituted for heavyweight peers in an AWT-based application without modifying the application or otherwise alerting the application to its presence. In other words, the AWT toolkit (not the AWT-based application) is modified so that it substitutes a reference to a lightweight peer whenever the AWT-based application calls for the display of an AWT control. Therefore, the present system and method overcomes the problems associated with conventionally mixed AWT and Swing APIs, while greatly improving application portability and reducing maintenance and upgrading costs. *See*, e.g., Specification, page 25, line 1 to page 30, line 2.

The article entitled *Introducing Swing* does just as the title implies – it briefly outlines the Swing architecture and the basic differences between Swing and AWT. However, *Introducing Swing* (otherwise referred to as IS-SUN) does not teach or suggest a system, method, or computer-readable storage device for graphical display of an object, where the system comprises a peer component, which is adapted to receive events pertaining to the object and route the events to a proxy component, which then associates the object with a graphic resource component and invokes methods of the graphic resource component to display the object, as taught in present claims 1, 9, and 17. In fact, the Office Action admits "IS-SUN... doesn't teach a proxy component which associates the object with the graphics resource component and invokes methods of the graphics resource component to display the object." (Office Action, page 2). The Office Action also admits that IS-SUN fails to teach "a peer component, adapted to receive events

pertaining to the object and route the events to the proxy." (Office Action, page 2). Consequently, IS-SUN does not teach or suggest all limitations of present claims 1, 9, and 17.

In addition, the article entitled *Mixing heavy and light components* (otherwise referred to as M-SUN) cannot be combined with IS-SUN to overcome the deficiencies therein. In particular, and as described in more detail below, M-SUN also fails to teach or suggest the aforementioned limitations of present claims 1, 9 and 17. However, statements in the Office Action suggest, "M-SUN teaches a method of implementing swing components similar to that of IS-SUN, but further teaches a proxy component that associates an object with a graphics resource component, and further displays the object, in that the proxy component is the swing class (see page 2, paragraph 2), and a peer component, adapted to receive events pertaining to the object and route the events to the proxy component, in that the peer component is the ancestor (see page 2, paragraph 2)." (Office Action, page 2). Further statements in the Office Action suggest that "it would have been obvious to one of ordinary skill in the art, having the teachings of IS-SUN and M-SUN before him at the time the invention was made to modify the swing component interface of IS-SUN to include the combinational properties as did M-SUN." (Office Action, page 3). As will be described in more detail below, however, IS-SUN and M-SUN cannot be combined or modified to teach or suggest the aforementioned limitations of the presently claimed case.

M-SUN discloses various rules or guidelines that should be followed when mixing heavyweight AWT components with lightweight Swing components, although such mixing is not generally recommended by M-SUN. *See*, e.g., page 2, paragraph 1, to page 3, paragraph 2 of M-SUN. For example, M-SUN states, "[b]ecause there's sometimes no alternative to mixing heavyweight and lightweight components, we have provided a few options in Swing to make a certain level of component-mixing possible." (M-SUN, page 3, paragraph 2). These options, or guidelines, include: "[d]o not mix lightweight (Swing) and heavyweight (AWT) components within a container where the lightweight component is expected to overlap the heavyweight one." (M-SUN, page 5, paragraph 2). This particular "guideline" was noted above and described in the presently claimed case as a problem that typically occurs when one attempts to mix lightweight and heavyweight components within the same container. Contrary to the presently claimed case, however, M-SUN provides no other solution to the problem other than sheer avoidance. M-SUN definitely fails to disclose the presently claimed system and method for displaying an object, where the system includes: 1) a proxy component for associating the object with a graphics resource component and invoking methods of the graphics resource component to display the object, and 2) a peer component for receiving events pertaining to the object and routing the events to the

proxy component, as taught in present claims 1, 9 and 17. Consequently, M-SUN does not teach or suggest all limitations of present claims 1, 9 and 17.

In light of the Examiner's suggestion that "the proxy component is the swing class" and "the peer component is the ancestor," the terms "class" and "ancestor" will be defined as they are used in the context of object-oriented programming. As described in the Specification, a "class" is a fundamental entity in an object-oriented programming language that possesses certain attributes (e.g., shape, color, size, location on screen, behavior, etc.). Once a class has been defined, one or more "objects" can be created as instances of the class. In addition, objects can be developed as modular building blocks, with subsequent objects referred to as children of parent objects. In this manner, a parent object may be considered a "class" of child objects, and the child objects may inherit the class attributes (i.e., the properties and methods) of the parent object, or "ancestor," from which they derive. In the embodiment of FIG. 8, for example, Component 32 may be considered a "class" of objects, where one of the objects in the class is Button 30. Component 32 may also be considered to be an "ancestor" of Button 30, since various methods and/or properties of Button 30 are derived from Component 32.

However, a "swing class" cannot be considered a "proxy component," as suggested by the Examiner, since a "swing class" merely defines the properties and methods (e.g., shape, color, size, location on screen, behavior, etc.) of a collection of swing objects, whereas a "proxy component" actually functions to associate an object (e.g., a button displayed in a GUI) with a graphics resource component (e.g., JButton 48) and to invoke the methods of the graphics resource component (e.g., run the program code contained within JButton 48) for displaying the object. A "swing class," in itself, does not and cannot function to associate an object with a graphics resource component, invoke the methods of the graphics resource component, or display the object. Therefore, the "swing class" described by M-SUN cannot be considered a "proxy component," as presently claimed.

In addition, though a "peer component" may be an "ancestor" of some other object, merely stating so provides no evidence of the peer component being adapted to receive events pertaining to the object and to route the events to a proxy component. Since M-SUN fails to disclose a "proxy component," the peer components described by M-SUN cannot be configured to route events to a non-existent proxy component. M-SUN simply fails to teach or suggest the aforementioned limitations of present claims 1, 9, and 17.

Since none of the cited art teaches or suggests the aforementioned limitations, the cited art cannot be combined in such a manner that teaches or suggests the aforementioned limitations of the presently claimed case.

Moreover, the cited art cannot be modified to teach or suggest the aforementioned limitation, since neither IS-SUN nor M-SUN suggests the desirability of doing so. The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990); MPEP 2143.01.

Unlike the presently claimed case, IS-SUN fails to mention the difficulties involved in mixing heavyweight AWT components and lightweight Swing components. Therefore, IS-SUN fails to suggest any desirability for providing a modified API that indirectly substitutes heavyweight peers with lightweight peers, in the manner described above and taught in the presently claimed case. Though M-SUN, on the other hand, does admit to some difficulties in mixing heavyweight AWT components and lightweight Swing components, M-SUN chooses to deal with the difficulties by avoiding the problem-causing situations altogether. By choosing avoidance, M-SUN fails to provide motivation for a system or method that actually solves the problem. Since IS-SUN and M-SUN each fail to provide motivation to teach or suggest the aforementioned limitations of the presently claimed case, IS-SUN and M-SUN cannot be modified to do so.

For at least the reasons set forth above, none of the cited art, either separately or in combination, teaches, suggests, or provides motivation for all limitations of independent claims 1, 9, and 17. Therefore, claims 1, 9, and 17, as well as claims dependent therefrom, are asserted to be patentably distinct over the cited art. Accordingly, removal of this rejection is respectfully requested.

CONCLUSION

This response constitutes a complete response to all issues raised in the Office Action mailed November 21, 2003. In view of the remarks traversing the rejections, Applicants assert that pending claims 1-17 are in condition for allowance. If the Examiner has any questions, comments, or suggestions, the undersigned attorney earnestly requests a telephone conference.

No fees are required for filing this amendment; however, the Commissioner is authorized to charge any additional fees, which may be required, or credit any overpayment, to Conley Rose, P.C. Deposit Account No. 03-2769/5468-07300.

Respectfully submitted,

Kevin L. Daffer
Reg. No. 34,146
Attorney for Applicant(s)

Conley Rose, P.C.
P.O. Box 684908
Austin, TX 78768-4908
(512) 476-1400
Date: February 23, 2004
JMF